Data Science Lab: Process and methods
Politecnico di Torino

Project report
Student ID: s277957

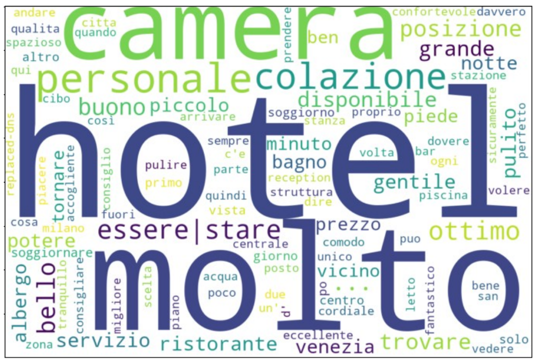
Exam session: Winter 2020

1. Data exploration (max. 400 words)

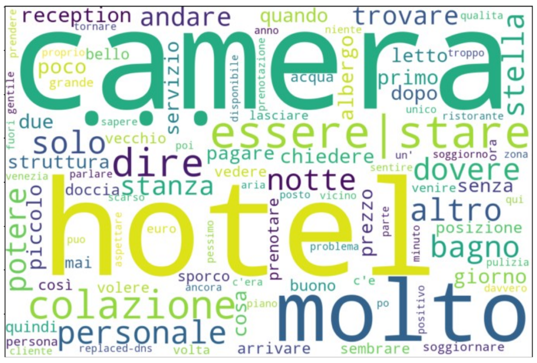
My data exploration step starts by counting how many positives and negatives comments are in the development.csv file. It contains 19532 positives and 9222 negatives comments that are respectively the 67.9% and 32% the of the dataset, which means unbalanced classes.

Secondly I search for important words that are related exclusively positive or negative comments, because I hope such words will help me in the tuning step. To do this I count the frequency of the words by means of CountVectorizer and I visualize the result by means of the WordCloud class.

Positive comments:



Negative comments:



Nevertheless there seems to be no difference in the frequency of the most used words in the respective labels. As a result I guess the weight assigned by the td-idf methods will be more usefull to the sentimental analysis.

2. Preprocessing (max. 400 words)

In order to extract the features from the documents I used the tf-idf methods that is implemented by sklearn library.
I choose as parameters:

- stop words from nltk.corpus.stopwords plus few words add by me, such as ‘@card@’ that represents numbers accordingly with the tokenizer used.
- strip_accents=’unicode’ that is a slightly slower method than ‘ascii’, but works on any characters.
- max_df = 0.7 used to ignored words too commons, because usually they don’t bring much information.
- min_df = 5 used to ignored words too rare, because very rare terms are not characterizing words.
- lowercase = True in order to convert all characters to lowercase before tokenizing.
- Tokenizer: Override the string tokenization step with my LemmaTokenizer2 class

The sparse matrix obtained consists in 10589 columns that are distinct words obtained throw the LemmaTokenizer2 class.

Inside the mentioned before class I use TreeTagger [1] to tokenize the commens and to lemmatize the words, because It works better than word_tokenizer plus ItalianStemmer from nltk library.

On the TreeTagger website is recommended to use it without stop words and with max_df = 0.3 and min_df = 5, however I am not confident without using stop words because I think the classifier will be more robust and generic if non-characteristic words are eliminated.

LemmaTokenizer2 class also takes care about some emoticons that can be relevant for sentimental analysis.

Finally I used a stop words list therefore I have set max_df=0.7 and min_df = 5 .

3. Algorithm choice (max. 400 words)

My algorithm choice is perform by a peace of code inspired by the web page Classifier Comparison[2] of sklearn.

First I fit every classifier that must be tested and evaluated them by means of f1_score() in order to understand which methods performs better with the default parameter passes at their constructors.

Results:

- Nearest Neighbors: 0.84
- Linear SVM: 0.96
- RBF SVM: 0.96
- Decision Tree: 0.81
- Random Forest: 0.56
- Naive Bayes: 0.88

Secondly after many test with the evaluation.csv I resolve to use the LinearSVC algorithm by sklearn, also because the advantage of linear models is that the feature weights directly correspond to the importance of the feature within the model. Thus it is easy to understand what the model has learned.[3]

More important words:

```
2.880360166334428 -> valdostano
2.453214762947466 -> accoccolare
2.256105896894239 -> anteguerra
```

```
Less important words:
-2.6498800628652974 -> testimoniare
-2.601847638080301 -> verdurine
-2.5278626272722233 -> pozzo
```

Finally I choose LinearSVC because its faster than SVC(kernel='linear') in my experiments.

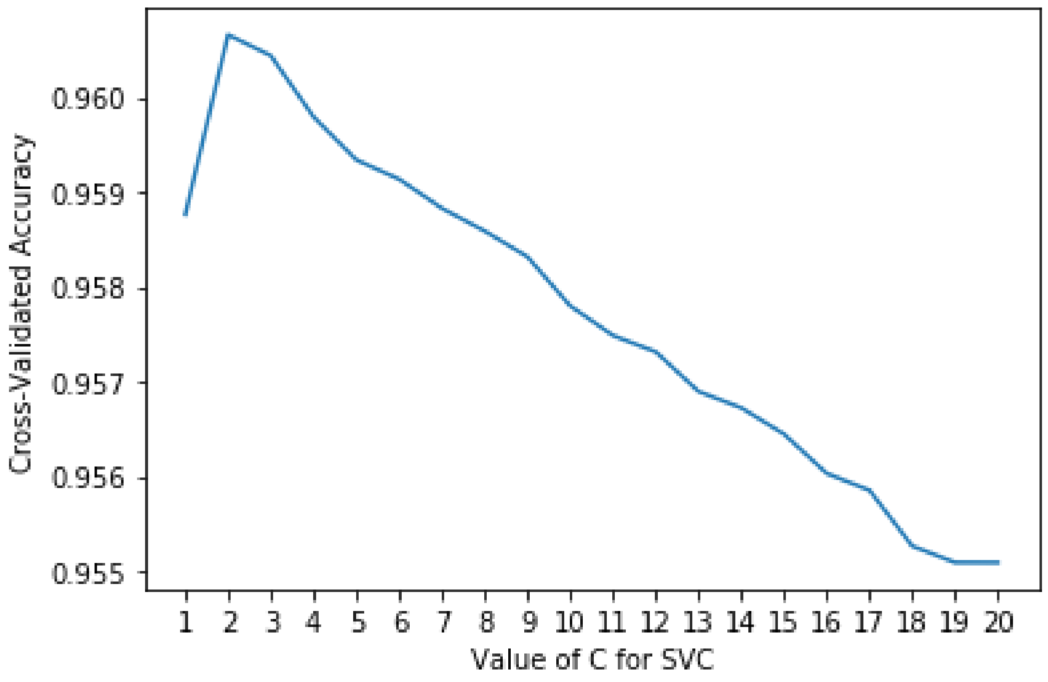
4. Tuning and validation (max. 400 words)

To build a robust and reliable classification model a cross validation on the data available in the development set is performed.

I decide to concentrate my search on the C parameter of the LinearSVC because I found that it is the one that varies the accuracy most after a few attempts on evaluation.csv.

I try with values ranging from 0.1 to 2 end cross validate the model over 3 partition of the development.csv in order to avoid overfitting.

The function make_score from sklearn.metrics helps me to validate the Support Vector Machines by the f1_score weighted.
I choose C = 0.2 as shown by the figures below:



Finally I make a pipeline to train the classification model over the entire development.csv.

5. References

[1] TreeTagger:
https://cis.uni-muenchen.de/~schmid/tools/TreeTagger/

[2] Classification Comparison:
https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

[3] Support Vector Machines:
https://scikit-learn.org/stable/modules/svm.html